### 一. Gateway介绍

***官网***: https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.2.RELEASE/reference/html/#configuration

Spring Cloud Gateway是Spring官方基于Spring 5.0，Spring Boot 2.0和Project Reactor等技术开发的网关，Spring Cloud Gateway旨在为微服务架构提供一种简单而有效的统一的API路由管理方式。Spring Cloud Gateway作为Spring Cloud生态系中的网关，目标是替代ZUUL，其不仅提供统一的路由方式，并且基于Filter链的方式提供了网关基本的功能，例如：安全，监控/埋点，和限流等。

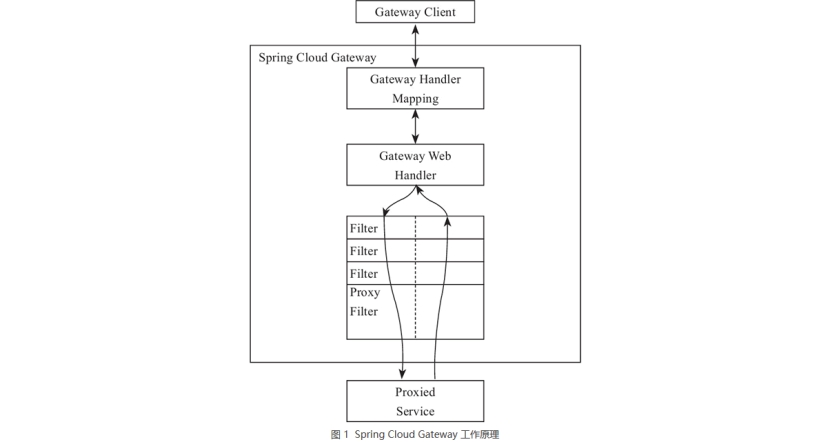
Spring Cloud Gateway 可以看做是一个 Zuul 1.x 的升级版和代替品，比 Zuul 2 更早的使用 Netty 实现异步 IO，从而实现了一个简单、比 Zuul 1.x 更高效的、与 Spring Cloud 紧密配合的 API 网关。

Spring Cloud Gateway 里明确的区分了 Router 和 Filter，并且一个很大的特点是内置了非常多的开箱即用功能，并且都可以通过 SpringBoot 配置或者手工编码链式调用来使用。

比如内置了 10 种 Router，使得我们可以直接配置一下就可以随心所欲的根据 Header、或者 Path、或者 Host、或者 Query 来做路由。

比如区分了一般的 Filter 和全局 Filter，内置了 20 种 Filter 和 9 种全局 Filter，也都可以直接用。当然自定义 Filter 也非常方便。

### 二. 工作原理



### 三. 与zuul的区别

#### 1. 开源组织

Spring Cloud Gateway 是 Spring Cloud 微服务平台的一个子项目，属于 Spring 开源社区，依赖名叫：spring-cloud-starter-gateway。  
  
https://spring.io/projects/spring-cloud-gateway  
  
Zuul 是 Netflix 公司的开源项目，Spring Cloud 在 Netflix 项目中也已经集成了 Zuul，依赖名叫：spring-cloud-starter-netflix-zuul。  
  
https://github.com/Netflix/zuul

#### 2. 底层实现

Zuul构建于 Servlet 2.5，兼容 3.x，使用的是阻塞式的 API，不支持长连接，比如 websockets。  
  
Spring Cloud Gateway构建于 Spring 5+，基于 Spring Boot 2.x 响应式的、非阻塞式的 API。同时，它支持 websockets，和 Spring 框架紧密集成，开发体验相对来说十分不错。

#### 3. 性能表现

WebFlux 模块的名称是 spring-webflux，名称中的 Flux 来源于 Reactor 中的类 Flux。Spring webflux 有一个全新的非堵塞的函数式 Reactive Web 框架，可以用来构建异步的、非堵塞的、事件驱动的服务，在伸缩性方面表现非常好。使用非阻塞API。 Websockets得到支持，并且由于它与Spring紧密集成，所以将会是一个更好的 开发 体验。

### 四.基本概念

#### 1.Route(路由)

网关的基本构建块,由一个ID,一个目标URI,一组断言和一组过滤器定义.如果断言为真,则路由匹配.

#### 2.Predicate(断言)

输入类型是一个ServerWebExchange.可以使用它来匹配来自HTTP请求的任何内容,例如headers或参数.

#### 3.Filter(过滤器)

Gateway中的filter分为两种类型的filter,分别为Gateway Filter 和Global Filter.过滤器将会对请求和响应进行修改处理.

### 五. GateWay使用

#### 1. 引入Gateway依赖

<!-- gateway 网关 -->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-gateway</artifactId>  
 </dependency>

#### 2. application.yml 配置

spring:  
 application:  
 name: gateway-9400  
 cloud:  
 ######## 配置注册中心信息 [Start] ########  
 nacos:  
 discovery:  
 server-addr: localhost:8848  
 #heart-beat-interval: 1000  
 #heart-beat-timeout: 30000  
 namespace: 019728c4-0a1a-4a7f-99e0-afd123aeae29  
 register-enabled: true  
 ######## 配置注册中心信息 [End] ########  
   
 ######## 配置Gateway信息 [Start] ########  
 gateway:  
 # default-filters 默认过滤器(应用到每个路由)  
 default-filters:  
 # 服务降级过滤器  
 - name: Hystrix  
 args:  
 name: fallback  
 fallbackUri: forward:/fallback  
 # 服务限流过滤器  
 - name: RequestRateLimiter  
 args:  
 # 允许用户每秒处理多少个请求  
 redis-rate-limiter.replenishRate: 1 # 每秒补充10个  
 # 令牌桶的容量，允许在一秒钟内完成的最大请求数  
 redis-rate-limiter.burstCapacity: 3 # 突发20个  
 redis-rate-limiter.requestedTokens: 1 # 每次请求消耗1个  
 rate-limiter: "#{@redisRateLimiter}" # 限流器  
 key-resolver: "#{@remoteAddrKeyResolver}" # 限流key|字段解析器  
   
 # 请求控制  
 routes:  
 - id: provider-user-9300 # 路由的id,没有规定规则但要求唯一  
 uri: lb://provider-user-9300 # lb表示启动gateway的负载均衡功能  
 filters: # 过滤器(只对当前路由起作用)  
 - name: RequestRateLimiter  
 args:  
 redis-rate-limiter.replenishRate: 10  
 redis-rate-limiter.burstCapacity: 20  
 key-resolver: "#{@userKeyResolver}"  
 predicates: # 断言,路径相匹配的进行路由  
 - Path=/user/login # 路径相匹配的进行路由  
  
 - id: provider-user  
 predicates:  
 - Path=/user/test  
 uri: lb://provider-user-9300  
 - id: consumer-order-9200  
 predicates:  
 - Path=/order/\*\*  
 uri: lb://consumer-order-9200  
 - id: provider-payment-9100  
 predicates:  
 - Path=/payment/\*\*  
 uri: lb://provider-payment  
 # 开启从注册中心动态创建路由的功能,利用微服务名进行路由  
 discovery:  
 locator:  
 enabled: true  
 ######## 配置Gateway信息 [End] ########

路由配置方式2

@Configuration  
public class GatewayConfig {  
   
 @Bean  
 /\*  
 \* @author liedu  
 \* @date 2020/7/14 13:51  
 \* @param [routeLocatorBuilder]   
 \* @return org.springframework.cloud.gateway.route.RouteLocator  
 \* @throw   
 \* @description 使用代码注入路由  
 \*/  
 RouteLocator customRouterLocator(RouteLocatorBuilder routeLocatorBuilder){  
 RouteLocatorBuilder.Builder routes = routeLocatorBuilder.routes();  
 routes.route("path\_baidu\_news",r -> r.path("/guonei").uri("http://news.baidu.com/guonei")).build();  
 return routes.build();  
 }  
  
}

#### 3.整合Hystrix进行服务降级

##### 1.引入Hystrix依赖

<!--hystrix-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
 </dependency>

##### 2.配置文件

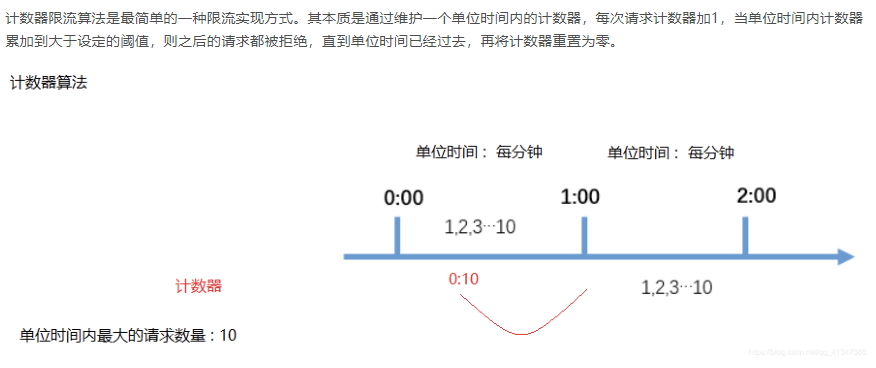
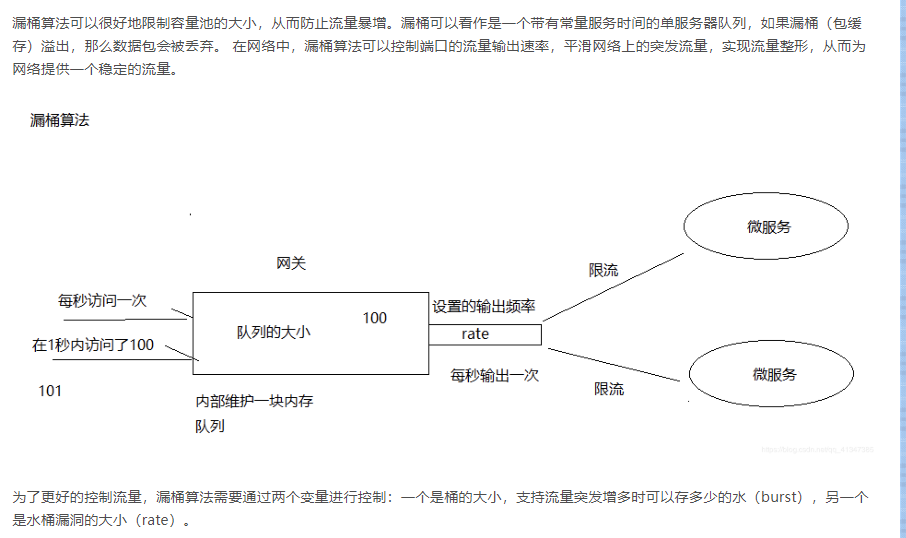
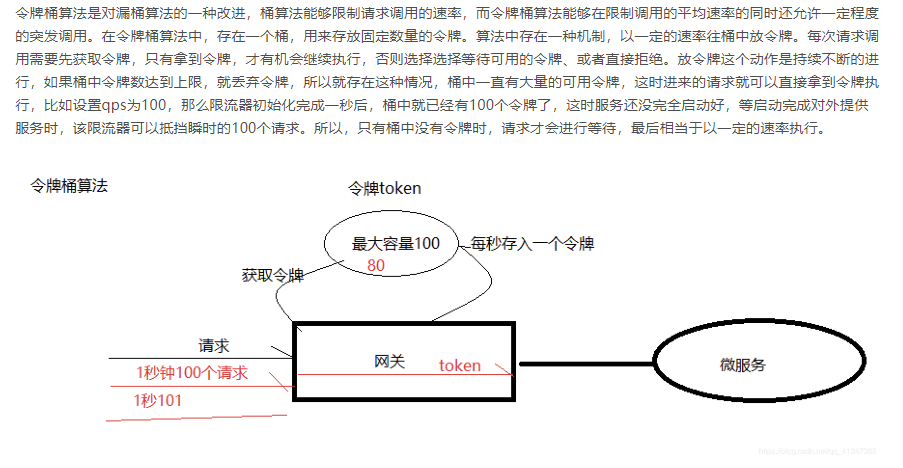
spring:  
 cloud:  
 gateway:  
 # gateway filter 默认的过滤器(应用到每个路由)  
 default-filters:  
 # 服务降级  
 - name: Hystrix  
 args:  
 name: fallback  
 fallbackUri: forward:/fallback  
hystrix:  
 command:  
 fallback:  
 execution:  
 isolation:  
 strategy: SEMAPHORE  
 thread:  
 #服务超时时间  
 timeoutInMilliseconds: 5000

##### 3.降级请求处理

* 新增Controller
* @RestController  
  public class FallBackController {  
   @GetMapping("/fallback")  
   public ResponseEntity fallback(){  
   return ResponseEntity.status(HttpStatus.SERVICE\_UNAVAILABLE).body("服务暂时不可用,请重试");  
   }  
  }
* 进行降级请求路由
* routes:   
   - id: fallback  
   predicates:  
   - Path=/fallback  
   uri: https://www.baidu.com

#### 4.请求限流

##### 1.限流算法(https://blog.csdn.net/qq*41347385/article/details/106548249?utm*medium=distribute.pc*relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase&depth*1-utm*source=distribute.pc*relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase)

* 计数器
* 
* 漏桶算法
* 
* 令牌桶算法
* 

##### 2.引入依赖

<!-- 限流 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  
 </dependency>

##### 3.application.yml配置

spring:  
 cloud:  
 gateway:  
 # gateway filter 默认的过滤器(应用到每个路由)  
 default-filters:  
 # 服务限流  
 - name: RequestRateLimiter  
 args:  
 # 允许用户每秒处理多少个请求  
 redis-rate-limiter.replenishRate: 1 # 每秒补充10个  
 # 令牌桶的容量，允许在一秒钟内完成的最大请求数  
 redis-rate-limiter.burstCapacity: 3 # 突发20个  
 redis-rate-limiter.requestedTokens: 1 # 每次请求消耗1个  
 key-resolver: "#{@remoteAddrKeyResolver}" # 限流key|字段解析器  
   
 redis:  
 host:   
 port: 6379  
 database: 1  
 password:

##### 4.新增限流规则类

import org.springframework.cloud.gateway.filter.ratelimit.KeyResolver;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Primary;  
import reactor.core.publisher.Mono;  
  
/\*\*  
 \* @author: liedu  
 \* @date: 2020/7/9 13:53  
 \* @description: gateway 限流配置类  
 \* @version: 1.0.0  
 \*/  
@Configuration  
public class RateLimiterConfig {  
  
 /\*  
 \* @author liedu  
 \* @date 2020/7/9 13:57  
 \* @param []  
 \* @return org.springframework.cloud.gateway.filter.ratelimit.KeyResolver  
 \* @throw  
 \* @description 通过访问ip进行限流  
 \*/  
 @Bean(value = "remoteAddrKeyResolver")  
 //@Primary  
 public KeyResolver remoteAddrKeyResolver(){  
 return exchange -> Mono.just(exchange.getRequest().getRemoteAddress().getAddress().getHostAddress());  
 }  
  
 /\*  
 \* @author liedu  
 \* @date 2020/7/9 15:07  
 \* @param []   
 \* @return org.springframework.cloud.gateway.filter.ratelimit.KeyResolver  
 \* @throw   
 \* @description 用户限流  
 \* 获取请求用户id作为限流key。  
 \*/  
 @Bean  
 public KeyResolver userKeyResolver() {  
 return exchange -> Mono.just(exchange.getRequest().getQueryParams().getFirst("userId"));  
 }  
  
 /\*  
 \* @author liedu  
 \* @date 2020/7/9 15:07  
 \* @param []  
 \* @return org.springframework.cloud.gateway.filter.ratelimit.KeyResolver  
 \* @throw  
 \* @description 接口限流  
 \* 获取请求地址的uri作为限流key。  
 \*/  
 @Bean  
 KeyResolver apiKeyResolver() {  
 return exchange -> Mono.just(exchange.getRequest().getPath().value());  
 }  
}

#### 5.使用Nacos做配置中心

##### 1.引入依赖

<dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>  
 </dependency>

##### 2.新建配置文件bootstrap.properties

配置文件优先级: 同一目录下 bootstrap.properties>bootstrap.yml>application.properties>application.yml  
  
spring.application.name=gateway-9400  
  
# nacos 配置中心地址  
spring.cloud.nacos.config.server-addr=127.0.0.1:8848  
# nacos配置中心命名空间  
spring.cloud.nacos.config.namespace=019728c4-0a1a-4a7f-99e0-afd123aeae29  
# nacos配置中心分组  
spring.cloud.nacos.config.group=GATEWAY  
# 配置文件类型  
spring.cloud.nacos.config.file-extension=yaml  
# 设置配置文件前缀  
spring.cloud.nacos.config.prefix=${spring.application.name}

##### 3.修改application.yml

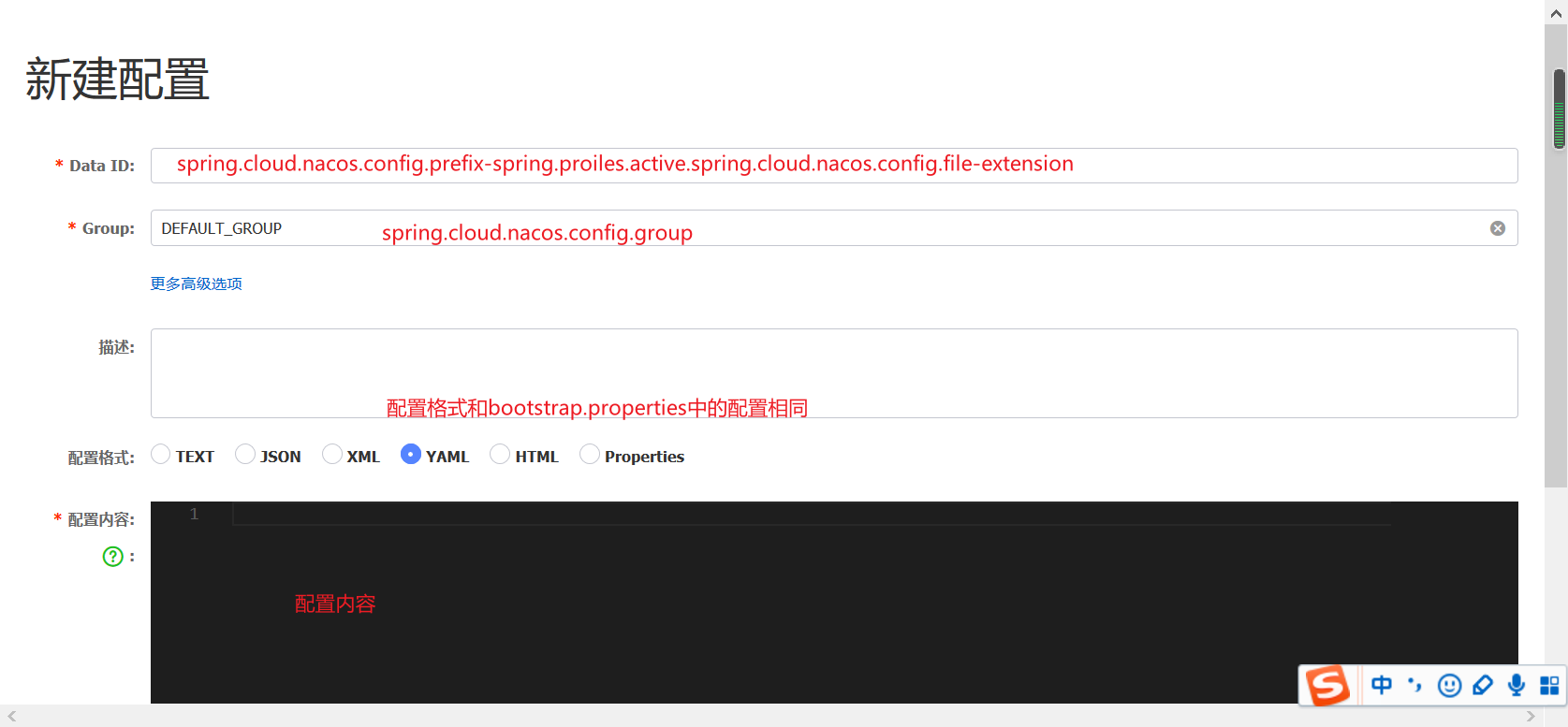
spring:   
 profiles:  
 active: dev

##### 4.nacos配置中心新增配置文件

Data ID: **spring.cloud.nacos.config.prefix**-*spring.profiles.active*.spring.cloud.nacos.config.file-extension

没有配置spring.cloud.nacos.config.prefix 默认为spring.application.name

没有配置spring.profiles.active 则默认为**spring.cloud.nacos.config.prefix**.spring.cloud.nacos.config.file-extension



发布配置文件

##### 5.修改主启动类

添加@RefreshScope 注解